Dynamic Formal Proof Presentation

Paul Jackson

Paul.Jackson@ed.ac.uk

School of Informatics
University of Edinburgh

TUPLE Student Conference
21st February 2024

Overview

Focus

- Challenges with reading and understanding proofs
  - Examples from Lean, Nuprl & Coq
  - Not assuming prior familiarity with any proof assistants
- Ideas on how to improve matters

Also

- Some other proof-assistant-related research directions

A procedural proof script (Lean)

```
theorem Nat.r2irrat_short :
  ¬ (∃ m n : ℕ,   Nat.Coprime m n ∧ m * m = 2 * n * n) :=
    by
  rintro ⟨m, n, hcp, heq⟩
  have h2divm : 2 ∣ m := by
    apply Nat.two_div_square
    simp [Nat.instDvdNat]
    use n * n
    linarith
  have h2divn : 2 ∣ n := by
    apply Nat.two_div_square
    simp [Nat.instDvdNat] at *
    rcases h2divm with ⟨cc, hcc⟩
    use cc * cc
    nlinarith
  rw [Nat.coprime_elim] at hcp
  have h2eq1 := hcp 2 (And.intro h2divm h2divn)
  linarith
```

A (semi) declarative proof (Lean)

```
theorem Nat.r2irrat_semi_decl :
 ¬(∃ m n : ℕ, Nat.Coprime m n ∧ m * m = 2 * n * n) := by
  rintro ⟨(m : ℕ), (n : ℕ), (hcp : Nat.Coprime m n),
                              (heq : m * m = 2 * n * n)⟩
  show False
  have h2divm : 2 ∣ m := by
    show 2 ∣ m
    apply Nat.two_div_square ; show 2 ∣ m * m
    simp [Nat.instDvdNat]    ; show ∃ c, m * m = 2 * c
    use n * n                ; show m * m = 2 * (n * n)
    linarith
  have h2divn: 2 ∣ n := by
    show 2 ∣ n
    apply Nat.two_div_square   ; show 2 ∣ n * n
    simp only [Nat.instDvdNat] ; show ∃ c, n * n = 2 * c
    ...
  have hhcp :  ∀ (c : ℕ), c ∣ m ∧ c ∣ n → c = 1 := by
    rw [Nat.coprime_elim] at hcp ; assumption
  have h2eq1 : 2 = 1 := hhcp 2 (And.intro h2divm h2divn)
  linarith
```

Subgoal tree made by a procedural proof script (Nuprl)

```
⊢ ¬(∃m,n:ℤ. CoPrime(m,n) ∧ m * m = 2 * n * n)
|
BY D 0 THENM ExRepD
|
1. m: ℤ
2. n: ℤ
3. CoPrime(m,n)
4. m * m = 2 * n * n
⊢ False
|
BY Assert ⌈2 | m⌉
|\
| ⊢ 2 | m
| |
| BY BLemma ‘two_div_square‘ THENM Unfold ‘divides‘ 0  THENM AutoInstConcl []
 \
  5. 2 | m
  |
  BY Assert ⌈2 | n⌉
  |\
  | ⊢ 2 | n
  | |
  | BY BLemma ‘two_div_square‘ THENM All (Unfold ‘divides‘) THENM ExRepD
  |     THENM Inst [⌈c * c⌉] 0 THENM RWO "6" 4 THEN Auto
   \
    6. 2 | n
    |
    BY RWO "coprime_elim" 3 THENM FHyp 3 [5;6]
    |
    3. ∀c:ℤ. c | m ⇒ c | n ⇒ c ∼ 1
    7. 2 ∼ 1
    |
    BY RWO "assoced_elim" 7 THENM D (-1) THEN Auto
```

Elision of repeated declarations, hypotheses, conclusions

With elision

```
...
|
1. m: ℤ
2. n: ℤ
3. CoPrime(m,n)
4. m * m = 2 * n * n
⊢ False
|
BY Assert ⌈2 | m⌉
|\
| ⊢ 2 | m
| |
| ...
 \
  5. 2 | m
  |
  ...
```

Without elision

```
...
|
1. m: ℤ
2. n: ℤ
3. CoPrime(m,n)
4. m * m = 2 * n * n
⊢ False
|
BY Assert ⌈2 | m⌉
|\
| 1. m: ℤ
| 2. n: ℤ
| 3. CoPrime(m,n)
| 4. m * m = 2 * n * n
| ⊢ 2 | m
| |
| ...
 \
  1. m: ℤ
  2. n: ℤ
  3. CoPrime(m,n)
  4. m * m = 2 * n * n
  5. 2 | m
  ⊢ False
  |
  ...
```

Focussing on a proof step

```
* top 1 2

1. m: ℤ
2. n: ℤ
3. CoPrime(m,n)
4. m * m = 2 * n * n
5. 2 | m
⊢ False

BY Assert ⌈2 | n⌉

1* ⊢ 2 | n

2* 6. 2 | n
   ⊢ False
```

Viewing more detail

Original proof

```
1. m: ℤ
2. n: ℤ
3. CoPrime(m,n)
4. m * m = 2 * n * n
5. 2 | m
⊢ 2 | n
|
BY BLemma ‘two_div_square‘
    THENM All (Unfold ‘divides‘)
    THENM ExRepD
    THENM Inst [⌈c * c⌉] 0
    THENM RWO "6" 4
    THEN Auto
```

Expanded proof

```
1. m: ℤ
2. n: ℤ
3. CoPrime(m,n)
4. m * m = 2 * n * n
5. 2 | m
⊢ 2 | n
|
BY BLemma ‘two_div_square‘
|
⊢ 2 | n * n
|
BY All (Unfold ‘divides‘)
|
5. ∃c:ℤ. m = 2 * c
⊢ ∃c:ℤ. n * n = 2 * c
|
BY ExRepD
|
5. c: ℤ
6. m = 2 * c
|
BY Inst ⌈c * c⌉ 0
|
⊢ n * n = 2 * c * c
|
BY RWO "6" 4
|
4. (2 * c) * 2 * c = 2 * n * n
|
BY Auto
```

A proof outline

```
*T root_2_irrat_over_int

⊢ ¬(∃m,n:ℤ. CoPrime(m,n) ∧ m * m = 2 * n * n)
|
BY Assume negation of goal and aim for proof by contradiction
|
1. m: ℤ
2. n: ℤ
3. CoPrime(m,n)
4. m * m = 2 * n * n
⊢ False
|
BY From hyp 4, deduce that 2 | m
|
5. 2 | m
|
BY From hyps 4 and 5, deduce that 2 | n
|
6. 2 | n
|
BY Observe that hyps 5 and 6 contradict hyp 3
```

Related work

- Alectryon generates proof displays with foldable intermediate goals for the Coq proof assistant.
  - Extended examples (e.g. Vol 1 of *Software Foundations*)

  (Clément Pit-Claudel)

- LeanInk extends Alectryon to work with Lean.
  (Niklas Bülow)

- Logique et démonstrations assistées par ordinateur – a Lean-based logic course
  - Click on grey rectangles to see formal goals
  - Click on French exposition lines in the *Démonstration* blocks to hide and reveal corresponding formal steps

  (Patrick Massot)

Proof Informalization

Taking formal proofs and automatically generating informal proofs with folded further details.

`https://kmill.github.io/`

(Patrick Massot & Kyle Miller)

Paperproof

Reorganises Lean proofs into Natural Deduction graphs

`https://github.com/Paper-Proof/paperproof`

(Evgenia Karunus & Anton Kovsharov)

Future challenges and opportunities

- Engineering the UI
- Languages & support for literate proof developments
- Viewing vs editing technologies
- Subgoal trees vs. subgoal stacks – metavariables
- Improving tracing of automatic procedures

See Dynamic Proof Presentation paper for further details.

Summary

Proof assistant have a variety of uses:

- Formal Verification (compilers, OS microkernels, security applications)
- Education (UG maths, computer science)
- Research (PL theory, maths)

WIth many, understanding proofs is important, but is often hard.

This talk has argued that

- better presentation of proof structure can help,
- dynamic presentation is better than static.

Are many other opportunities to dynamically present and explain

- formulas and expressions,
- tactics,
- libraries.

Further information on Proof Assistant Research

Kinds

- Applications
- Foundations
- Systems Engineering
- Use of Machine Learning
  (IMO Grand Challenge)

Conferences

- ITP (Interactive Theorem Proving)
- CPP (Certified Programs and Proofs)
- CICM (Intelligent Computer Mathematics)

Workshops

For Lean, Coq, Agda, Isabelle, ACL2.